Learning Practical Communication Strategies in Cooperative Multi-Agent Reinforcement Learning

Diyi Hu DIYIHU@USC.EDU
Chi Zhang ZHAN527@USC.EDU
Viktor Prasanna PRASANNA@USC.EDU
Bhaskar Krishnamachari BKRISHNA@USC.EDU
University of Southern California, Los Angeles, CA 90007

Editors: Emtiyaz Khan and Mehmet Gönen

Abstract

In Multi-Agent Reinforcement Learning, communication is critical to encourage cooperation among agents. Communication in realistic wireless networks can be highly unreliable due to network conditions varying with agents' mobility, and stochasticity in the transmission process. We propose a framework to learn practical communication strategies by addressing three fundamental questions: (1) *When*: Agents learn the timing of communication based on not only message importance but also wireless channel conditions. (2) *What*: Agents augment message contents with wireless network measurements to better select the game and communication actions. (3) *How*: Agents use a novel neural message encoder to preserve all information from received messages, regardless of the number and order of messages. Simulating standard benchmarks under realistic wireless network settings, we show significant improvements in game performance, convergence speed and communication efficiency compared with state-of-the-art.

Keywords: Multi-agent Reinforcement Learning; Wireless communication

1. Introduction

In Multi-Agent Reinforcement Learning (MARL), communication plays a key role in facilitating knowledge sharing and collaboration. The information in agents' messages alleviates the limitation of *partial observation*. Communication leads to better exploration on the huge state and action space, and thus improves the training quality and convergence speed.

While there have been numerous works in the literature to improve communication in MARL, most are based on *unrealistic* modeling of the wireless network environment (*e.g.*, assuming perfect transmission). In practical applications, however, transmission can fail due to many factors such as limited bandwidth, signal path loss and fading, medium contention, interference, *etc.* Moreover, in applications involving navigation (*e.g.*, Search-And-Rescue (SAR) Queralta et al. (2020), fire fighting Haksar and Schwager (2018), battlefield defense Nguyen et al. (2019)), link conditions are dynamic as they vary with agents' mobility and relative positions. It remains an open problem to learn a practical policy by addressing such complexity in realistic wireless communications Hu et al. (2022).

The following challenges exist to train communicating agents in a realistic wireless environment. The first challenge, *environment coupling*, is due to the mutual influence between

the game environment and the wireless environment. On the one hand, agents' mobility in the game environment leads to dynamic network connectivity and agents' communication actions have significant impact on medium contention and signal interference. On the other hand, changes in the wireless environment can affect agents' actions in the game environment. For example, when an agent generating an important message is far away from other agents, it may intentionally approach others to increase the receivers' signal strength. The second challenge, *channel stochasticity*, means it is a non-deterministic process whether an agent can successfully receive the message from others. As described in the previous paragraph, many practical factors can result in failed transmission. The stochasticity not only increases the complexity of the environment, but also makes it more challenging for agents to extract useful information.

Proposed work We propose a framework, `LAUREL`, to Learn prActical commUnication strategies in cooperative Multi-Agent REinforcement Learning. Our framework optimizes three fundamental aspects of communication (*i.e.*, when, what and how) without unrealistic simplifications on the wireless network. To address the environment coupling challenge, we propose to delay the message sending time by one step so as to align agents' interactions in the game and wireless environments. The alignment solves the issue of training non-differentiability in Kim et al. (2019). More importantly, it leads to a "meta-environment" abstraction, where the Markov Decision Process (MDP) controlling the agents' behaviors can be reformulated, and both the action and observation spaces can be expanded. Specifically, we augment the agents' observation by important network measurements, so that they can understand and predict the dynamics due to coupling. Then we end-to-end train a practical communication strategy with neither pre-defined schedule nor prior knowledge on the wireless condition (unlike Kim et al. (2019); Sukhbaatar et al. (2016)). As a result, agents learn when to send messages based on both the content relevance and the wireless channel conditions. To further improve `LAUREL` under high channel *stochasticity*, we propose a novel neural architecture to encode the received messages. The encoder takes as input *any* number of messages and theoretically guarantees an asymptotically lossless encoding process. Finally, we generalize `LAUREL` with popular backbone MARL algorithms, and build both the on- and off-policy variants in a plug-and-play fashion. On standard benchmarks, we achieve significantly better performance with respect to game performance, convergence rate and communication efficiency, compared with state-of-the-art methods.

2. Related Work

CommNet (Sukhbaatar et al., 2016) and BiCNet (Peng et al., 2017) propose to learn the communication contents in the MARL system to boost performance. However, both works perform all-to-all communication at each step, under the assumption of *ideal wireless network* conditions (*i.e.*whatever messages sent out will be successfully received by all agents). In terms of message aggregation under such assumption, a simple mean or sum function is used. TarMAC (Das et al., 2019) and MAGIC (Niu et al., 2021) later introduce attention-based mechanisms to improve message aggregation.

Recent works consider *resource constrained wireless networks*, which improves the ideal model by considering limited bandwidth. When multiple agents simultaneously send k' messages, the environment ensures $\min\{k, k'\}$ messages to be successfully received by all

agents (where k is a manually configured constant characterizing the bandwidth limit). VBC (Zhang et al., 2019) limits the variance of the message and only those with large variance are exchanged during execution. SchedNet (Kim et al., 2019) generates an importance weight for the message, and exactly k most important messages are exchanged in each step. RMADDPG (Wang et al., 2019) adopts a recurrent multi-agent actor-critic framework which limits the number of communication actions during training. Note that these works still assume perfect transmission, and ignore many critical factors in a realistic wireless environment. All works above assume at least one of the following: (1) Dedicated and perfect wireless channels exist between any pair of agents. Any transmitted message can be delivered to all other agents successfully. (2) Message importance is the only factor when deciding "when to send a message". (3) Channel condition is fixed and known in advance to allow scheduling agents' communication with a fixed budget. In comparison, our framework does not have any of the above assumptions and considers realistic wireless channels.

3. Preliminaries

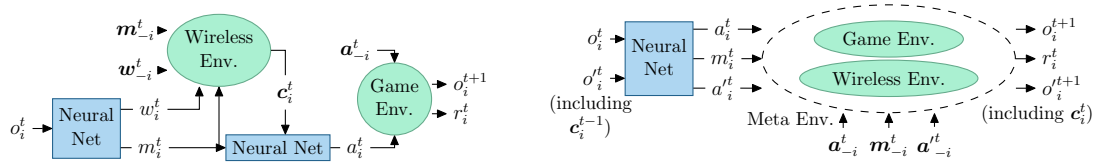
Dec-POMDP Agents solve a problem formulated as a Decentralized Partially-Observable Markov Decision Process (Dec-POMDP), which is represented by $(\mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \boldsymbol{\Omega}, \mathcal{O}, \mathcal{R}, \gamma)$. $\mathcal{S}$, $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\Omega}$ are the state space, joint action space, and joint observation space for the N agents, respectively. At step t, each agent $i \in N$ takes action a_i by following policy π_{θ_i}. The joint action $\boldsymbol{a}$ leads to state transition captured by function $\mathcal{T}(s'|s, \boldsymbol{a}) : \mathcal{S} \times \boldsymbol{\mathcal{A}} \times \mathcal{S} \to [0, 1]$. The agent also draws local observation $o_i \in \Omega_i$, with joint observation probability $P(\boldsymbol{o}|s', \boldsymbol{a}) = \mathcal{O}(\boldsymbol{o}, s', \boldsymbol{a})$, where $\boldsymbol{o}$ is the joint observation. Meanwhile, agents receive a joint reward $r_t = \mathcal{R}(s, \boldsymbol{a})$. The goal is to find an optimum policy $\boldsymbol{\pi_{\theta^*}}$ that maximizes the expected long-term reward (*i.e.*, return), with discount factor $\gamma \in [0, 1]$: $J = \mathbb{E}_{\boldsymbol{\pi_\theta}}\left[\sum_{t=1}^{\infty} \gamma^t r_t\right]$.

Communication Environment In MARL applications with mobile agents (*e.g.*, SAR), agents altogether formulate a Mobile Adhoc network (MANET) (Ismail and Ja'afar, 2007) to communicate. In a large and complicated terrain, network conditions vary with agents' mobility. *e.g.*, there may be obstacles of different materials blocking the signal propagation. Under popular wireless protocols (*e.g.*, p-CSMA Gai et al. (2011)), before agents send, they contend to access the medium with certain probability to avoid collision. The signal strength of a transmitted data packet may be affected by path loss, attenuation, interference with other packet signals, *etc.* In particular, for path loss, log-normal fading model is widely used to capture the Radio Signal Strength (RSS) variation over the spatial domain. For interference, it happens when receiver is in range of multiple senders at the same time. When SINR is below a given threshold, the packet cannot be decoded correctly. In such a case, a broadcast message cannot be received by all agents. See Appendix A and C.1 for details.

4. Method

4.1. When: MDP Re-Formulation

Use subscript "$-i$" to denote variables from all agents other than i. Communication can facilitate exchange of *current* observations. A natural design, followed by most existing



(a) Original: non-differentiable training (b) New: alignment after lagging

Figure 1: Two ways of agent-environment interaction

works (Foerster et al., 2016; Sukhbaatar et al., 2016; Singh et al., 2019), is to send and receive a message within the same step. In Figure 1(a), at the beginning of step t, agent i makes local observation o_i^t on the game environment. From o_i^t, a neural network generates a message m_i^t with weight w_i^t measuring message importance / relevance. Then based on w_i^t and $\boldsymbol{w}_{-i}^t$, the messages m_i^t and $\boldsymbol{m}_{-i}^t$ contend to go through the wireless channel. Since some transmissions may fail, we use $\boldsymbol{c}_i^t$ to denote the messages successfully received by i. Next, agent i's own message and all its received ones go through another neural network to generate the next action a_i^t. Finally, all agents interact with the game environment via a_i^t and $\boldsymbol{a}_{-i}^t$. The game environment returns the current reward r_i^t and next observation o_i^{t+1}.

A major drawback of the Figure 1(a) setting is training non-differentiability. When optimizing policy π_θ via back-propagation, the gradients of the parameters θ need to flow from a_i^t back to o_i^t. However, the wireless environment lies in the middle of step t. The mapping implemented by the realistic wireless channel, $\boldsymbol{m}^t, \boldsymbol{w}^t \rightarrow \boldsymbol{c}_i^t$, is non-differentiable (Kim et al., 2019) (due to "stochasticity" in transmission).

Communication Lagging We propose "communication lagging" to make training differentiable. Denote a' as communication actions and o' as wireless observations (see Section 4.2). As shown in Figure 1(b), an agent observes at the beginning of step t. Yet it does not generate the message m_i^t until the end of step t. After lagging, the "message-wireless environment" and "agent-game environment" interactions happen simultaneously (enclosed by dotted circle). Thus, the two environments are *aligned*. Lagging leads to the following tradeoffs: (1) *Ease of training*: the aligned environments enable end-to-end policy training, as the gradients can flow backward from the end of step t (*i.e.*, a_i^t and a'^t_i) to the beginning of step t (*i.e.*, o_i^t and o'^t_i). We thus develop a general framework (Section 4.4) supporting any existing MARL algorithm on top of realistic wireless environment. (2) *Staleness of messages*: action a_i^t is generated with stale information $\boldsymbol{c}_i^{t-1}$ since the step-$(t-1)$ messages are received at step t. Such staleness may only have slightly negative impact when the step duration is short in practice (Zhang et al., 2019).

Meta-Environment We abstract a "meta-environment" from aligned game and wireless environments. Meta-environment is the *new* game environment: any algorithm on the original game environment can be directly applied on the meta-environment. To define agents' interaction with the meta-environment, we reformulate the Markov Decision Process (MDP) with expanded state and action spaces: $(\mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \boldsymbol{\Omega}, \mathcal{O}, \mathcal{R}, \gamma)$. Denote $\mathcal{S}$ as the state of the meta-environment. The *augmented* action space $\boldsymbol{\mathcal{A}}$ consists of the game actions $\boldsymbol{\mathcal{A}}^{\mathcal{T}}$ and communication actions $\boldsymbol{\mathcal{A}}^{\mathcal{C}}$. *i.e.*, $\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{A}}^{\mathcal{T}} \times \boldsymbol{\mathcal{A}}^{\mathcal{C}}$. The *augmented* observation space $\boldsymbol{\Omega}$

consists of the game observations $\mathbf{\Omega^T}$ and wireless observations $\mathbf{\Omega^C}$. *i.e.*, $\mathbf{\Omega} = \mathbf{\Omega^T} \times \mathbf{\Omega^C}$. $\mathcal{T}$ and $\mathcal{O}$ are defined on the meta-state and augmented observation and action spaces.

Learning Communication Actions Under the reformulated MDP, agents can learn a policy on communication actions to decide "when to communicate". The simplest is the binary action for "send / no-send": $a_i^C \in \mathcal{A}_i^C = \{0, 1\}$. Other examples of learnable $a_i^{\mathcal{C}}$ include: (1) *Transmission power* P_t: larger P_t improves the received signal strength at the cost of higher probability of interference. (2) *Medium contention probability* p: larger p increases the probability of securing the medium by suppressing other agents' chance.

4.2. What: Observation Augmentation

Efficiently exploring the state space $\mathcal{S}$ and action space $\mathcal{A}$ relies on suitable observations, especially considering the complexity in wireless environment. However, since agents may be far away, it is infeasible for them to directly observe the full network condition (*i.e.*, the wireless environment is partially observable). To address the issue, we first augment the observation space $\mathcal{O}$ by observation / measurement $o_i^{\mathcal{C},t}$ on the wireless environment. *i.e.*. for each agent, we concatenate the wireless and game observations $o_i^t = o_i^{\mathcal{T},t} \| o_i^{\mathcal{C},t}$. Then we let the message sent by i contain i's hidden state $\boldsymbol{h}_i^t$, where $\boldsymbol{h}_i^t$ embeds information of o_i^t. When j receives from i, agent j predicts the current network condition. If such condition is believed as poor, j may suppress its sending action.

Network Measurements Common factors resulting in failed transmission include limited bandwidth, noises, signal attenuation due to obstacles and packet collision. The network condition is complicated: some of the above factors are related to the wireless environment (*e.g.*, limited bandwidth), some are related to the game environment (*e.g.*, obstacles), and others are even related to agents' policies (*e.g.*, collision due to simultaneous sendings). Agents can understand the network condition from various types of network measurements. For brevity, we now only illustrate the benefits of augmenting observation via *Radio Signal Strength* (RSS) in detail. When agent i receives the message from j, it measures P_s and also infers j's position (thus distance d between i and j). When agent i receives more messages from other agents across multiple steps, it essentially collects many (P_s, d) data points. The agent can then learn the function $P_s = g(d)$ corresponding to the path loss model of the particular wireless environment. With g, when an agent knows (or predicts) the position of others, it immediately knows the corresponding RSS if it sends a message. Ignoring collisions, the agent knows if its message can be successfully received or not, *even before it sends out the message*. Such knowledge clearly helps with a better policy on communication actions. In addition, RSS information can also reveal terrain information on the game environment. For instance, if the measured P_s is significantly lower than the estimated $g(d)$, the agent detects an obstacle on the transmisson path. Communication then serves as a form of long-range sensing on the terrain, and helps agents better plan their future movements.

4.3. How: Message Encoder

Stochasticity in Communication An agent does not know (1) who will send a message at what time, (2) whether the message will have enough signal strength to be successfully

received, (3) at what order messages will arrive, (4) what the message will contain. For point 1, agents in non-deterministic RL algorithms randomly select communication actions from a probability distribution. For point 2, many factors (*e.g.*, path loss, noise, interference) can cause packet loss. For point 3, due to medium contention and varying network latency, multiple messages received by a single agent in one step can arrive in an arbitrary order. For point 4, for the hidden state embedding contained in a message, the vector elements can take arbitrary numerical values representable by a floating point number.

Message Encoder We design a neural architecture to encode received messages by addressing all the stochasticity. In the following, we consider a single step, and thus omit superscript (t). Denote vector $\boldsymbol{m}_{ij} \in \mathbb{R}^l$ as the message from agent j to i. Denote $\mathcal{N}_i$ as the set of agents successfully sending to i. Cardinality of $\mathcal{N}_i$ is from 0 (when i receives no message) to $n-1$ (when i receives from all other agents). Define $c_i = \{\boldsymbol{m}_{ij} : j \in \mathcal{N}_i\}$ as the set of messages received by i. The encoder performs a function $\Phi(c_i)$ to map the set of messages to a vector in $\mathbb{R}^{d'}$. Φ should satisfy the following two properties.

Permutation invariance This property addresses the stochasticity 3 above. Since we define the input to Φ as an *un-ordered* set c_i, we need to preserve a well-known property of such a set function, namely permutation invariance (Zaheer et al., 2017). By definition, for any $c_i = \{\boldsymbol{m}_{i1}, \ldots, \boldsymbol{m}_{ik}\}$ and any permutation ρ on the indices $\{1, \ldots, k\}$, a permutation invariant function satisfies

$$\Phi(\{\boldsymbol{m}_{i1}, \ldots, \boldsymbol{m}_{ik}\}) = \Phi\left(\left\{\boldsymbol{m}_{i\rho(1)}, \ldots, \boldsymbol{m}_{i\rho(k)}\right\}\right) \tag{1}$$

We show an example in our scenario. Suppose agent 1 receives messages from agents 2, 3 and 4. Imagine two sequences of message arrival, $(2, 3, 4)$ and $(3, 4, 2)$. The encoder should not care about the sequence and should generate the same output for both cases. Thus, $\Phi(\{\boldsymbol{m}_{12}, \boldsymbol{m}_{13}, \boldsymbol{m}_{14}\}) = \Phi(\{\boldsymbol{m}_{13}, \boldsymbol{m}_{14}, \boldsymbol{m}_{12}\})$.

Injectiveness This property addresses stochasticity 1, 2, and 4. To preserve all information in the encoding process, we should have

$$c_i = c_i' \quad \Leftrightarrow \quad \Phi(c_i) = \Phi\left(c_i'\right) \tag{2}$$

This way, agent i generates a unique encoding for c_i so that it knows exactly the status of *all* the agents who successfully sends a message to it. For a vector function, injectiveness is easy to satisfy. For example, a linear mapping $\Phi'(\boldsymbol{m}') = \boldsymbol{W}\boldsymbol{m}'$ is injective when $\boldsymbol{W}$ has rank l. However, it is non-trivial to ensure injectiveness of Φ whose input is a *set* with non-fixed size. In the literature (Sukhbaatar et al., 2016; Singh et al., 2019; Kim et al., 2019), a common way to implement Φ is by first aggregating the messages into a single vector and then encode the aggregated vector by a vector function. *e.g.*, sum aggregation $\Phi(c_i) = \Phi'\left(\sum_{j\in\mathcal{N}_i} \boldsymbol{m}_{ij}\right)$. Unfortunately, such a Φ is not injective since regardless of Φ', we can find cases where $c_i \neq c_i'$ yet $\sum_{j\in\mathcal{N}_i} \boldsymbol{m}_{ij} = \sum_{j\in\mathcal{N}_i'} \boldsymbol{m}'_{ij}$.

Different from the literature, we implement the message encoder as follows,

$$\text{Message encoder:} \qquad \Phi(c_i) = \textstyle\sum_{j\in\mathcal{N}_i} \texttt{MLP}(\boldsymbol{m}_{ij}) \tag{3}$$

where $\texttt{MLP}(\cdot)$ is a Multi-Layer Perceptron. All received messages go through a shared MLP. Our encoder theoretically satisfies the above two properties and preserves all information.

Theorem 1 *There exists a set of parameters for the encoder architecture defined by Equation 3, such that Φ is both permutation invariant and injective.*

Proof Please see Appendix B. ■

Relation with GNNs Our encoder can be seen as a single-layer Graph Neural Network tailored for real-life communication stochasticity. Correspondingly, "sum of MLPs" of LAUREL's Equation 3 is the GNN aggregation function, which preserves the theoretical properties while being simple to implement in practice. Communication design of many related works can also be related with GNNs. For example, TarMAC (Das et al., 2019) performing "weighted mean" of messages can be seen as a single-layer Graph Attention Network (Veličković et al., 2018); IC3Net (Singh et al., 2019) performing "unweighted average" is equivalent to GraphSAGE (Hamilton et al., 2017). Other related works (*e.g.*, MAGIC (Niu et al., 2021)) deploy multi-layer GNNs that require multi-hop communication in each step. Such communication can be too expensive under constraints of realistic wireless networks, and thus we restrict LAUREL to the single-layer design. Importantly, for all the above related works, their GNN encoders satisfy permutation invariance but *not injectiveness*. As a result, there can be significant performance drop due to the aforementioned stochasticity.

4.4. General Framework

We propose a ***general*** framework to learn practical communication strategies, since our techniques can be applied to various MARL training algorithms in a plug-and-play fashion.

Algorithm 1: Example off-policy LAUREL

Init. all neural networks and replay buffer $\mathcal{D}$
for *episode*$= 1$ *to* M **do**
  Init. observation $\boldsymbol{o}$; init. messages $\boldsymbol{c}$ to all 0
  for $t = 1$ *to end of episode* **do**
    Receive msg c_i^{t-1}; get network measurement $\boldsymbol{o}_i^{\mathcal{C},t}$
    $\phi_i^t \leftarrow$ Encode message by $\Phi_i\left(c_i^{t-1}\right)$
    $a_i^t, m_i^t, Q_i^t \leftarrow \epsilon$-greedy $\arg\max_{a_i} Q_i^*\left(o_i^t, \phi_i^t, a_i\right)$
    Take action $\boldsymbol{a}^{\mathcal{T},t}$ in game environment
    Send $\boldsymbol{\phi}^t$ by $\boldsymbol{a}^{\mathcal{C},t}$ in wireless environment
    Observe reward r^t; transit to state $\boldsymbol{s}^{t+1}$
    Add transition to replay buffer $\mathcal{D}$
  end
  Randomly sample trajectory from $\mathcal{D}$
  Update θ_Φ, $\theta_Q, \theta_{\text{mix}}, \theta_Q^{'}, \theta_{\text{mix}}^{'}$
end

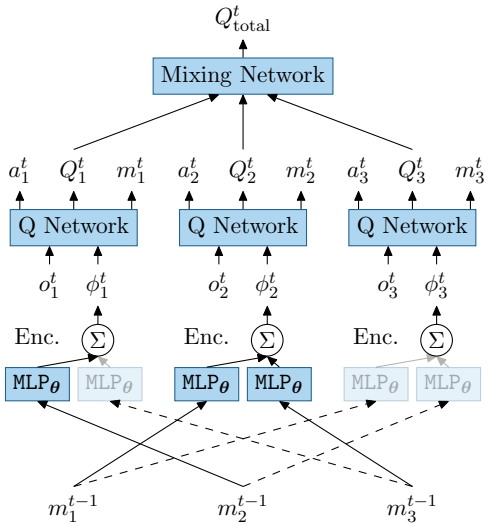


Figure 2: Example architecture of off-policy LAUREL

For illustration, we describe an example implementation based on value function factorization (Sunehag et al., 2018; Rashid et al., 2018), which can be trained in the ***off-policy***

manner. As shown in Figure 2, Three neural networks interact with each other: *message encoder*, Q *network* and *mixing network*. The message encoder follows the Section 4.3 design. The Q network estimates the Q value for each individual agent. The mixing network combines each agent's Q value to calculate a global Q for all agents. This design follows the "centralized training, decentralized execution" paradigm (Kraemer and Banerjee, 2016). During centralized training, the mixing network optimizes the policy via maximizing the global Q value. Parameter sharing is usually adopted assuming using an environment simulator. During decentralized execution, each agent outputs its next action with its own Q network alone, without the mixing network. The inputs to i's Q network include i's current augmented observation o_i^t and the encoded message vector ϕ_i^t. The output is the augmented action a_i^t and the Q value of $\boldsymbol{a}_i^t$. The inputs to the mixing network are the output Q of each Q network. The output is the global Q value for the current step t.

Algorithm 1 shows the training algorithm. Since we use GRU (Chung et al., 2014) in Q network, we perform gradient update at the end of rollout episode instead of every step (as in Wang et al. (2019)). In line 13, we perform gradient descent from the loss

$$\mathcal{L}(\theta) = \sum_t \left[(y_t^{\text{tot}} - Q_{\text{tot}}(\boldsymbol{o}, \boldsymbol{c}, \boldsymbol{a}; \theta))^2 \right] \tag{4}$$

where $y_t^{\text{tot}} = r + \gamma \max_{\boldsymbol{a}'} Q_{\text{tot}}(\boldsymbol{o}', \boldsymbol{c}', \boldsymbol{a}'; \theta')$, $\boldsymbol{c}$ is the received message. Q_{tot} is the full architecture in Figure 2 with parameter θ. $(\boldsymbol{o}, \boldsymbol{c}, \boldsymbol{a}, r, \boldsymbol{o}', \boldsymbol{c}')$ is transition-t in the sampled trajectory.

We can similarly integrate ***on-policy*** algorithms into our framework. In Section 5, we evaluate REINFORCE (Williams, 1992) based algorithms. The gradient function is $\nabla_\theta J(\theta) = \sum_i \sum_t [\nabla_\theta \log \pi_\theta(a_i|o_i, c_i) G_t]$, where G_t is the sampled returns. Other variants (*e.g.*, based on Actor-Critic) can also be derived similarly. We omit the details for brevity.

5. Experiments

5.1. Setup

Wireless Environment Following most existing works (Foerster et al., 2016; Sukhbaatar et al., 2016; Kim et al., 2019), we let agents perform single-hop broadcast. We implement a 1-hop mobile network without Access Points (AP) as in Flushing et al. (2014). For communication model, we follow "log distance path loss". We model interference as the receiver hearing multiple signals in range. We also consider background noise and attenuation due to obstacles. For communication protocol, we implement the slotted p-CSMA (Gai et al., 2011). Agents following the protocol perform fully distributed execution without the need of a centralized controller. See Appendix C.1 for details.

MARL Algorithms & Hyperparameters We evaluate both on- and off-policy designs.

Off-policy We compare three designs. The first is QMix Rashid et al. (2018), the state-of-the-art value decomposition based MARL training algorithm. The second enhances the communication part of the first design, by integrating the TarMAC message aggregation function (Das et al., 2019) into the original QMix training. The third is the `LAUREL` version of QMix, whose implementation follows Section 4.4 and Algorithm 1. Note that TarMAC is the state-of-the-art design performing attention-based message aggregation.

On-policy We again compare three designs. Among them, CommNet (Sukhbaatar et al., 2016) and IC3Net (Singh et al., 2019) are two state-of-the-art MARL algorithms with learned communication schemes. The `LAUREL` variant of IC3Net follows the description at the end of Section 4.4. All three are trained with REINFORCE (Williams, 1992).

We conduct most experiments with off-policy models due to its significantly higher sample efficiency and faster convergence time. See Appendix C.2 for training hyperparameters. `LAUREL`is open source on Github [1].

5.2. Predator Prey

"Predator-Prey" is a standard MARL benchmark (Kim et al., 2019; Singh et al., 2019; Zhang et al., 2019). In this game, m predators and 1 prey are initially randomly placed in an $n \times n$ grid world (denoted as PP_n). In one step, each predator can take one of the five possible *game actions*: moving to an adjacent grid or staying still. *Reward* is given when a predator catches the prey by moving to the prey's position. The game *terminates* when all predators catch the prey. For *observation*, each predator has vision v.

We further propose variants of the PP environment to better simulate realistic applications (*e.g.*, SAR with complex field terrain). In the vanilla setting used by Kim et al. (2019); Sukhbaatar et al. (2016); Singh et al. (2019), the grid world is flat and does not contain any obstacles. In the advanced setting, we introduce k obstacles. An obstacle is specified by size ℓ and can be either horizontal or vertical. Obstacles affect both the game and wireless environments, since they block movement of agents and also have an attenuation effect on the wireless signals passing them. To initialize each episode, we randomly generate the obstacle orientation and positions. We denote such an environment as PP_n^{Obs}. For all experiments, we set grid size $n = 10$. There are 3 agents, each with vision $v = 0$ (*i.e.*, similar to Singh et al. (2019), agents only see the grid it is currently in), number of obstacles $k = 1$ and obstacle size $\ell = 9$. Detailed parameter settings of wireless environment can be found in Appendix C.2.

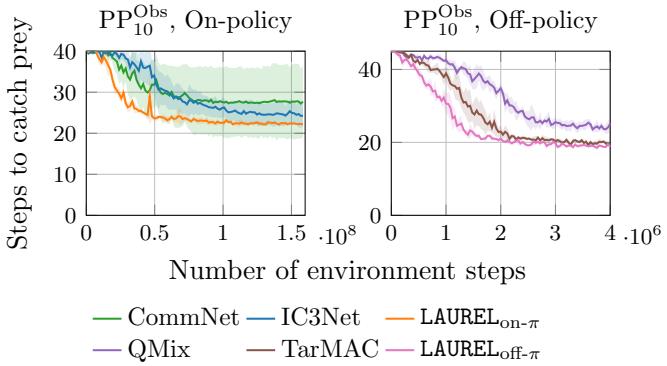


Figure 3: Comparison with state-of-the-art methods

Comparison with State-of-the-Art `LAUREL` uses RSS as the wireless observation. From Figure 3, we observe that for both the on-policy and off-policy algorithms, `LAUREL` significantly shortens the number of steps to catch the prey. In addition, the variances of the

1. https://github.com/ANRGUSC/LAUREL

LAUREL curves are very small. For CommNet, the model hard-codes an all-to-all communication scheme: each step, all agents broadcast messages. In a realistic wireless network environment, sending more messages can reduce the number of successfully received messages due to increased chance of collision and interference. Therefore, it can be hard for the algorithm always performing broadcasting to stably learn a good policy – as reflected by the large variance of the CommNet curve. For IC3Net, the performance is better than CommNet since IC3Net has gated communication which mutes unimportant messages. LAUREL further improves upon IC3Net due to its intelligence in all the "when", "what" and "how" aspects. For the off-policy comparisons, QMix converges to a policy with significantly more steps. This shows the importance of communication. For LAUREL and TarMAC, both converge to similar number of steps. However, LAUREL converges faster and with smaller variance. Note that in the Lumberjacks environment (Section 5.3), TarMAC fails to learn a good policy while LAUREL can.

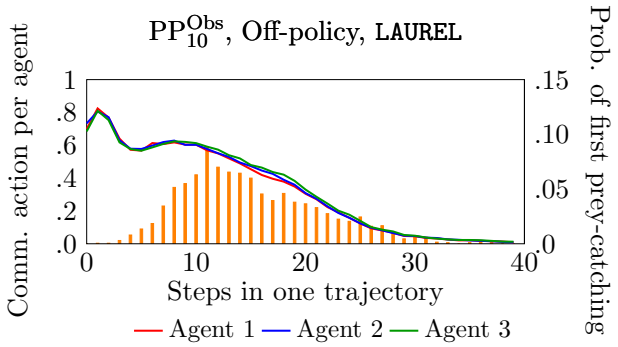


Figure 4: Average communication action in one trajectory

Learned Communication Strategy We analyze the communication scheme learned by our framework. We first answer "*at which steps of the trajectory the agents are more likely to send messages*". In Figure 4, we use the same configuration of LAUREL as Figure 3. Once the training converges, we freeze the model and evaluate it on 2000 trajectories. We record two metrics: (1) the binary communication action a_i^t of each agent i at each step t, and (2) the first step t' where at least one predator catches the prey. The left vertical axis (corresponding to the curves) of Figure 4 records the average a_i^t over the 2000 trajectories. The right vertical axis (corresponding to the bars) records the probability of t'. We have the following observations: (1) The three curves almost overlap with each other since agents are homogeneous. (2) At the initial few steps, the agents are more likely to communicate. This allows agents to know others' initial positions, as well as to probe the wireless and game environments. The sooner they know such information, the better they collaborate. (3) The agents are also more likely to communicate when they catch the prey (as can be seen from the similar shape of the curves and the bars after step 10). After the prey-catching agent informs others of the prey's position, the other agents can directly approach the prey following the shortest path.

Adapting to Complicated Wireless Environment We further study the effect of bandwidth limit on agents' communication behaviors. In Figure 5, we consider two wireless

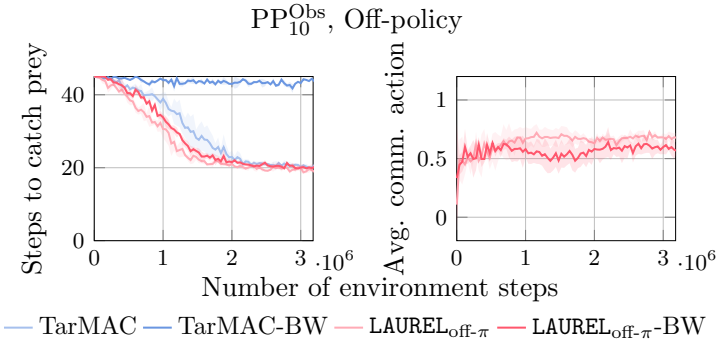


Figure 5: Communication adapted to limited bandwidth

environments: one with more bandwidth resources and the other with fewer bandwidth resources (achieved by reducing the number of time slots in slotted p-CSMA (Gai et al., 2011)). All other parameters remain as default. The curves marked by "-BW" correspond to those evaluated in the bandwidth reduced environment. We observe the different behaviors of TarMAC and `LAUREL`. In the right plot, we plot the average communication action α. *e.g.*, when $\alpha = 0.5$, each agent has 50% probably of sending a message at each step. Since TarMAC agents always send a message in each step ($\alpha = 1$), we do not plot their communication curves. We observe that: (1) The performance of TarMAC is very sensitive to the wireless environment. When the wireless environment becomes more complicated, the total number of steps to catch the prey increases significantly. (2) For `LAUREL`, making the wireless environment more complicated only results in slightly slower convergence. From the right plot, we observe that `LAUREL` successfully adapts its policy to the changed environment by reducing the number of communications.

Ablation Study We show how our encoder improves the quality of the learned policy. In Table 1, we compare three different encoding architectures, with all other configurations equivalent. For the MLP encoder, suppose each message is a length d vector. For 3 agents, the input $\boldsymbol{e}$ is a length-$3d$ vector. The subvector $[\boldsymbol{e}]_{id:(i+1)d}$ is filled with i's message if message from agent i is received, otherwise, 0s. Then $\boldsymbol{e}$ is fed into a 2-layer MLP to generate the encoded vector. For the "Avg" encoder, we first aggregate the received raw messages by vector mean, and then feed the aggregated vector into a 2-layer MLP. Clearly, the proposed encoder based on Equation 3 leads to significantly better agents' performance.

Table 1: Comparison on number of steps

Enc.	MLP	Avg	Eq. 3
$\text{PP}_{10}^{\text{Obs}}$	24.03±5.12	21.24±0.98	18.71±0.58

5.3. Lumberjacks

Lumberjacks (Albrecht and Ramamoorthy, 2015) is a multi-resource spatial coverage problem (Kamra and Liu, 2020). In a grid world, p lumberjacks cooperate to chop q trees. Each tree has a "strength" k, meaning the tree will be chopped when at least k agents are adjacent to it. We set $k = 2$ for all trees. The agent obtains reward $r_1 = 0.05$ and $r_2 = 0.5$ for observing and chopping the tree. Step penalty $r_3 = -0.1$ is used to encourage agents

to chop trees within minimum number of steps. Similar to PP, each lumberjack agent can move to its adjacent grid or stay still in a step. We modify the original setting (Koul, 2019) by assigning trees signal attenuation $\delta = 4.5$. The game is terminated once all trees are chopped or the max number of steps is reached. Denote $\text{LJ}_n^{p,q}$ as a Lumberjacks game within $n \times n$ grid.

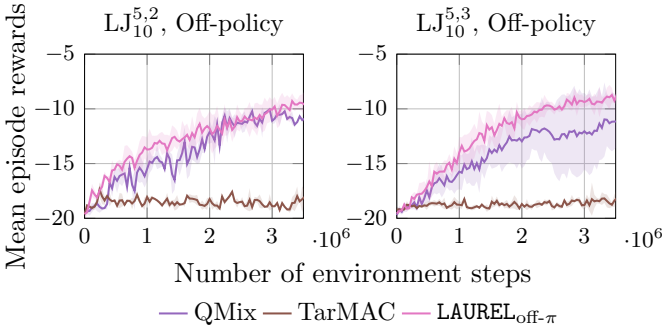


Figure 6: Comparison with state-of-the-art methods

Figure 6 compares the convergence of reward between LAUREL and state-of-the-art methods. Both environments contain 5 agents. In $\text{LJ}_{10}^{5,2}$ with 5 agents and 2 trees, QMix achieves similar reward as LAUREL. In $\text{LJ}_{10}^{5,3}$, LAUREL achieves significantly higher reward than QMix. In both environments, TarMAC fails to learn a good policy. We conclude the following. When there are many agents and few trees, collaboration is less critical – Even if each agent searches for a tree on its own and then waits there, there is still a high chance that two agents land adjacent to the same tree by coincidence. This is why the vanilla QMix performs well in $\text{LJ}_{10}^{5,2}$. However, when we increase the number of trees in the environment, collaboration becomes more critical, and the communication strategy becomes important. So we see a performance drop in QMix in contrast to a performance boost in LAUREL. Finally, the poor performance of TarMAC shows it is important to consider the complicated wireless environment in the algorithm design. Otherwise, *communication can be even more harmful than no communication at all*, as shown by the TarMAC-QMix comparison.

6. Conclusion

We have proposed a general framework to learn practical multi-agent communication strategies. Our techniques comprehensively address the fundamental aspects of communication, "when", "what" and "how", with theoretical and empirical justifications. The two implementations of our framework (on- / off-policy) significantly improve the agents' performance, especially in complicated environments. In the future, we plan to integrate the communication information in the centralized value factorization process. By capturing the learned communication structure in the mixer, we expect further performance improvement in terms of the quality of the policy.

Acknowledgments

This work was supported in part by ARL under Cooperative Agreement W911NF-17-2-0196.

References

Stefano V Albrecht and Subramanian Ramamoorthy. A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems. *arXiv:1506.01170*, 2015.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL http://arxiv.org/abs/1412.3555.

Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Mike Rabbat, and Joelle Pineau. Tarmac: Targeted multi-agent communication. In *ICML*, 2019.

Eduardo Feo Flushing, Michal Kudelski, Luca M Gambardella, and Gianni A Di Caro. Spatial prediction of wireless links and its application to the path control of mobile robots. In *IEEE SIES*, 2014.

Jakob Foerster, Yannis M Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *NeurIPS*, 2016.

Yi Gai, Shankar Ganesan, and Bhaskar Krishnamachari. The saturation throughput region of p-persistent csma. In *Information Theory and Applications*. IEEE, 2011.

David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *ICLR*, 2017.

Ravi Haksar and Mac Schwager. Distributed deep reinforcement learning for fighting forest fires with a network of aerial robots. In *IROS*, 2018.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 1989. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(89)90020-8.

Diyi Hu, Chi Zhang, Viktor Prasanna, and Bhaskar Krishnamachari. Intelligent communication over realistic wireless networks in multi-agent cooperative games. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, pages 1627–1629, 2022.

Datuk Prof Ir Ishak Ismail and Mohd Hairil Fitri Ja'afar. Mobile ad hoc network overview. In *IEEE APACE*, 2007. doi: 10.1109/APACE.2007.4603864.

Nitin Kamra and Yan Liu. Differentiable approximations for multi-resource spatial coverage problems. 2020.

Daewoo Kim, Sangwoo Moon, David Hostallero, Wan Ju Kang, Taeyoung Lee, Kyunghwan Son, and Yung Yi. Learning to schedule communication in multi-agent reinforcement learning. In *ICLR*, 2019.

D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Anurag Koul. ma-gym: Collection of multi-agent environments based on openai gym. https://github.com/koulanurag/ma-gym, 2019.

Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 2016.

Thanh Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. Multi-agent deep reinforcement learning with human strategies. In *IEEE ICIT*, 2019.

Yaru Niu, Rohan Paleja, and Matthew Gombolay. Multi-agent graph-attention communication and teaming. In *IFAAMAS*, 2021.

Peng Peng, Quan Yuan, Ying Wen, Yaodong Yang, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games. *arXiv:1703.10069*, 2017.

J. P. Queralta, J. Taipalmaa, B. C. Pullinen, V. K. Sarker, T. N. Gia, H. Tenhunen, M. Gabbouj, J. Raitoharju, and T. Westerlund. Collaborative multi-robot systems for search and rescue: Coordination and perception. *arXiv:2008.12610*, 2020.

Tabish Rashid, Mikayel Samvelyan, Christian Schröder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *ICML*, 2018.

Amanpreet Singh, Tushar Jain, and Sainbayar Sukhbaatar. Individualized controlled continuous communication model for multiagent cooperative and competitive tasks. In *ICLR*, 2019.

Sainbayar Sukhbaatar, arthur szlam, and Rob Fergus. Learning multiagent communication with backpropagation. In *NeurIPS*, 2016.

P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Leibo, K. Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *IFAAMAS*, 2018.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Rose E Wang, Michael Everett, and Jonathan P How. R-maddpg for partially observable environments and limited communication. *ICML*, 2019.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 1992. ISSN 0885-6125. doi: 10.1007/BF00992696.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *ICLR*, 2018.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *NeurIPS*, 2017.

Sai Qian Zhang, Qi Zhang, and Jieyu Lin. Efficient communication in multi-agent reinforcement learning via variance based control. *NeurIPS*, 2019.

Appendix A. Log Distance Path Loss with Fading

$$P_r = P_t - K_{\text{ref}} - 10\eta \log_{10} \frac{d}{d_0} + \psi \tag{5}$$

where P_r is the received power in dBm, P_t is the transmission power in dBm, K_{ref} is the loss at reference distance d_0, η is the path loss component depending on the propagation medium, and ψ is a log normal variable for multi-path fading. This equation describes the change of signal strength during the propagation.

Appendix B. Proofs

Proof [Proof of Theorem 1] To prove permutation invariance, note that vector addition is commutative. So for any permutation ρ, the sum of the sequence $\left(\texttt{MLP}\left(\boldsymbol{m}_{i\rho(1)}\right), \ldots, \texttt{MLP}\left(\boldsymbol{m}_{i\rho(k)}\right)\right)$ is always the same.

For injectiveness, we recall the conclusion from Lemma 5 of Xu et al. (2018).

Lemma 2 *Assume the space of messages, $\mathcal{M}$, is countable. There exists $f : \mathcal{M} \to \mathbb{R}^{d'}$, s.t. $\sum_{j \in \mathcal{N}_i} f\left(\boldsymbol{m}_{ij}\right)$ is unique for each $c_i \subset \mathcal{M}$.*

Then we only need to show an MLP can express the function f specified by the above lemma. This is a direct consequence of the universal approximation theorem (Hornik et al., 1989). In other words, there exists a set of MLP parameters such that $\Phi\left(c_i\right) = \sum_{j \in \mathcal{N}_i} \texttt{MLP}\left(\boldsymbol{m}_{ij}\right) = \sum_{j \in \mathcal{N}_i} f\left(\boldsymbol{m}_{ij}\right)$, where $c_i = c_i' \Leftrightarrow \Phi\left(c_i\right) = \Phi\left(c_i'\right)$. So Φ is injective.

Note that the requirement on the input space $\mathcal{M}$ being countable is automatically satisfied in realistic applications, where the message $\boldsymbol{m}$ is quantized to fixed bit widths. ■

Appendix C. Details in Experiments

C.1. Communication Protocols

Slotted p-CSMA When an agent intends to transmit a packet, it first randomly choose a counter in the range of "counter window size". When counter is down to 0, it senses the medium. If the medium is free (*i.e.*, sensed signal strength lower than the threshold Θ_f), the agent transmits with probability p. Otherwise, it chooses a random counter again and repeats the process.

Arbitration of successful receiving We use SINR (signal-to-interference-plus-noise ratio). The receiver agent will sense the arrived packet's signal strength as θ. Denote θ' as the interfering signal strength at the agent if exist, N as the background noise, Θ_r as the SINR threshold. If $\frac{\theta}{\theta'+N} < \Theta_r$, the packet cannot be decoded correctly. In other words, there are three cases for the receiving signal strength: (1) If $\theta > \Theta_r \times (\theta' + N)$, then the agent successfully receives the packet; (2) If $\Theta_f \leq \theta \leq \Theta_r \times (\theta' + N)$, then the agent knows there is a packet coming, but it cannot successfully receive the contents; (3) If $\theta < \Theta_f$, then the agent doesn't even know there is a packet arrived.

C.2. Parameters

Default settings used in Section 5 Obstacle attenuation $\delta_0 = 4.5$; noise $N = -95$dBm; RSS threshold $\Theta_f = -78$dBm; SINR threshold $\Theta_r = 15$ dB for PP_n^{Obs} and 20dBm for others; contention probability in p-CSMA protocol $p = 0.3$; counter window size in p-CSMA protocol $W = 15$ time slots.

Training Hyperparameters We repeat all experiments 5 times with different random seeds. We report the average performance and variance of the metrics. For the on-policy algorithms, we use a LSTM cell with hidden dimension 128 in the policy net. For the off-policy algorithms, we use a GRU cell with a 2-layer MLP of hidden dimension 128 as the Q-value network, and 3-layer MLP with hypernetwork (Ha et al., 2017) as the mixing network. For both versions of `LAUREL`, we use a 2-layer MLP with hidden dimension 128 to implemented the message encoder according to Equation 3. All algorithms are trained with Adam optimizer (Kingma and Ba, 2015) with learning rate 0.0005.